LEVEL II

[illegible]

12

AD A1[illegible]

The Complexity of Manipulating Hierarchically Defined Sets of Rectangles

Jon Louis Bentley
Departments of Computer Science
and Mathematics
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213
U.S.A.

Thomas Ottmann
Institut fur Angewandte Informatik
und Formale Beschreibungsverfahren
Universitat Karlsruhe
Postfach 6380, D-7500 Karlsruhe
West Germany

April 1981

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

CMU-CS-81-109

The Complexity of Manipulating Hierarchically Defined Sets of Rectangles.

Jon Louis Bentley[1]
Departments of Computer Science
and Mathematics
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213
U.S.A.

Thomas Ottmann[2]
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
Universität Karlsruhe
Postfach 6380, D-7500 Karlsruhe
West Germany

Abstract

Algorithms that manipulate sets of rectangles are of great practical importance in VLSI design systems and other applications. Although much theoretical work has appeared recently on the complexity of rectangle problems, it has assumed that the inputs are given as a list of rectangles. In this paper we study the complexity of rectangle problems when the inputs are given in a hierarchical language that allows the designer to build large designs by replicating small designs. We will see that while most of the problems are NP-hard in the general case, there are O(N lg N) algorithms that process inputs obeying certain restrictions.

[1] This work was supported in part by the United State Office of Naval Research under Contract N00014-76-C-0370.

[2] This work was partially supported by the Deutsche Forschungsgemeinschaft DFG.

1. INTRODUCTION

Algorithms that manipulate sets of rectangles in the plane are of great interest to practitioners and theoreticians alike. Practical applications of such algorithms arise in such areas as computer graphics, architectural design systems and VLSI (Very Large Scale Integrated Circuitry) design systems. These problems also have great appeal from a purely mathematical viewpoint: the problems are quite natural and easy to pose, yet the solutions often have a rather subtle structure.

Much theoretical work has been done recently on rectangle problems; we will return to a brief survey of that work in Subsection 3.1. Almost all of the work, though, has concentrated on rectangle sets that are defined by merely listing all the rectangles they contain. Although this is quite appropriate for many applications, for many others this model is terribly inaccurate. VLSI circuits, for example, are usually not specified by simply listing their components, but rather are described in a hierarchical design language that allows a designer to build big designs in an orderly way from small designs.

One way to process the hierarchical designs is merely to throw away their structure and treat them as though they were given as sets of rectangles. This can be quite costly, however, because such designs can describe an exponentially large number of rectangles.

With this motivation several researchers have recently begun to investigate the problem of dealing directly with a hierarchical description of a set of geometric objects; see, for example, Hon [1980] and Whitney [1980]. The approach that they have taken, however, is a solution-oriented strategy that has not led them to investigate many of the theoretical questions that arise in this endeavor.

The purpose of this paper is to provide a sound theoretical basis for the important problem of manipulating hierarchical descriptions of geometric objects. In Section 2 we will define the rectangle problems we will study and then consider the various forms in which their input might be given. In Section 3 we investigate the complexities of the (seven) problems when their inputs are given in (three) different representations. Finally, implications of the results and directions for further research are studied in Section 4.

2. PROBLEMS

In order to state a geometric problem precisely and to measure its complexity, we must specify the language which is used to describe an instance of the problem. In this section we will first give a list of rectangle problems and their applications using geometric terms in their intuitive meaning. We will then introduce a hierarchical language which allows us to describe the input and output of each problem and to measure its size.

The first problems in our list are intersection problems: we are given a set of rectangles in the plane (with sides parallel to the coordinate axes) and we ask for intersections among these rectangles or with other given objects. Two rectangles are said to intersect if the interior of their intersection contains at least one point. Thus, the intersection includes both proper edge intersection and the inclusion of one rectangle within another, but not two rectangles that touch only at the border.

1. *Report Intersecting Pairs*

Input: A set of rectangles.

Output: A list of all intersecting pairs of rectangles.

Application: This is an important task in the "geometry engine" underlying most VLSI design rule checkers; see Haken [1980] or Hon [1980].

2. *Intersection Question*

Input: A set of rectangles.

Output: Yes, if there is at least one pair of intersecting rectangles in the set; otherwise no.

Motivation: This is a more mathematically tractable version of problem 1.

3. *Intersection with a Query Object*

Input: A set of rectangles, and

a) a point, or

b) a line (parallel to x- or y-axis), or

c) a rectangle

Output: Yes, if

a) the query point lies in the interior of at least one rectangle in the set;

b) the line crosses the interior of at least one rectangle in the set;

c) the query rectangle intersects at least one rectangle in the set; otherwise no.

Applications: a) This allows an interactive user to point to a rectangle and have the system retrieve it for him.

b) This arises in routing (see Lauther [1980]).

c) This tests the validity of placing a rectangle.

4. *Northernmost Rectangle Below a Line*

Input: A set of rectangles and a horizontal line $y = y_o$. Note that an important special case occurs when y_o is greater than all y-values in the set.

Output: A northernmost rectangle in the set below the given line.

Application: This is related to the routing problem of Lauther [1980].

5. *Measure and Perimeter Problems*

Input: A set of rectangles.

Output: The

a) measure or

b) perimeter of their union; i.e. the total area covered by at least one rectangle in the set.

Motivation: These problems were raised in a theoretical context by Klee [1977].

6. *Connectedness Problems*

Input: A set of rectangles.

Output: a) Yes, if their union is a connected set, and, no, otherwise

b) the number of connected components in the union

c) a list of all connected components in the union.

Applications: Connectedness is a central notion in VLSI circuit extraction.

7. *Equality and Subset Testing of Rectangle Sets*

Input: Two descriptions D_1 and D_2 of sets of rectangles.

Output: Yes, if

a) the set denoted by D_1 is equal to the set denoted by D_2

b) the set denoted by D_1 contains the set denoted by D_2

otherwise no.

Observe that these problems are meaningful only if a language to describe sets of rectangles has been fixed.

We will now specify a very simple hierarchical language for describing sets of rectangles. This Hierarchic Input Language (HIL) may be considered as a proper subset of the Symbolic Layout Language defined in Mead and Conway [1980, Section 4.3] and of CIF (see Section 4.5 of the above).

The HIL language describes sets of rectangles as collections of boxes. Box commands describe each of these boxes by specifying the x,y- coordinates of the lower-left corner and its length and height. We may assume that all coordinates and lengths are nonnegative integer multiples of the layout unit λ. For instance, the command

```
Box (o,o) (1,1)
```

describes the unit square with lower-left corner at the origin. The general form of a Box command is

```
Box (X Coord,Y Coord)(Width,Height)
```

In HIL we can also define symbols that denote sets of rectangles. A symbol definition is a symbol number, followed by zero or more attributes and a list of Box and Draw commands. A Draw command has the form

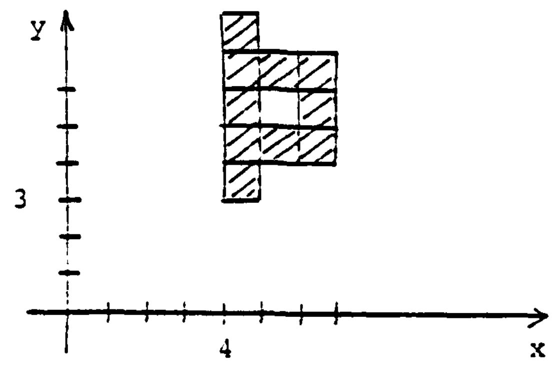
Draw < symbol number > at < point >

<symbol number> must be the number of a previously defined symbol, and <point> is a pair of (x,y) coordinates. This command describes the placement of the symbol denoted by the symbol number at the given point. To make this more precise we consider every defined symbol as "hooked" to the origin. Thus, drawing a symbol at a point means to place its origin on that point.

We give an example of a symbol definition with empty lists of attributes:

```
1 :  Box (0,0)(3,1)
2 :  Box (0,0)(1,3)
3 :  Box (4,3)(1,5)
     Draw 1 at (4,4)
     Draw 1 at (4,6)
     Draw 2 at (6,4)
```

Symbol 3 is as follows:



We consider it as hooked to the origin. The set of rectangles denoted by symbol number 3 consists of two copies of box 1, one copy of box 2 and one copy of the box in symbol 3 placed in the plane as shown by the above figure.

We now introduce the notion of a <u>bounding rectangle</u>, or BR, of a symbol: A bounding rectangle of a symbol is any rectangle that includes all boxes in the set of rectangles denoted by the symbol. A BR is specified by the coordinates of its lower-left corner and its width and height. Thus, the rectangle with lower-left corner at (3,3), length 4 and height 5 is a BR of the above defined symbol 3 (note that it is not the minimal BR).

HIL allows us to augment a symbol definition with an attribute which specifies a BR (not necessarily the minimum BR) of the symbol. Thus, a symbol definition has the general form:

$$\begin{aligned}\langle\text{symbol number}\rangle\text{:}\ &\{\text{BR}\langle\text{point}\rangle(\langle\text{length}\rangle,\langle\text{height}\rangle)\}_0^1\\ &\{\langle\text{box command}\rangle\}_0^*\\ &\{\langle\text{draw command}\rangle\}_0^*\end{aligned}$$

We will assume w.l.o.g. that the set of rectangles (i.e., boxes,) defined by a HIL file is the symbol denoted by the largest symbol number, which we will often call the root symbol. Symbol calls, i.e. symbol numbers occurring in draw commands of a symbol definition, may nest. That is, the definitions of the called symbols may contain calls of other, previously defined symbols etc. Thus, we can associate to every symbol definition a call graph which reflects the hierarchical structure of the symbol definition. The above definition of the symbol 3 has the following call graph:

$$\begin{tikzcd} 3 \arrow[d, bend right] \arrow[d, bend left] \arrow[dr, bend left] & \\ 1 & 2 \end{tikzcd}$$

Note that the call graph is always a directed acyclic graph, which we abbreviate as a DAG.

We define the length of the HIL description of a symbol as the total number of Box- and Draw commands occurring in the definition of the symbol. Thus, the length of the above HIL definition is 6. Observe that we do not count the length of the numbers (symbol numbers, coordinates, etc.) occurring in the definition. (We will return to this point in Subsection 3.2).

One easily observes that the same set of rectangles may have very different HIL descriptions, whose lengths may differ by an exponential factor. We may give a description of a set of N rectangles in the plane not using the hierarchical structure of HIL at all by writing N Box commands in the definition, one for each rectangle. This description of a set of rectangles is obviously equivalent to the usual assumption made in much previous work on geometric problems, namely, that the set of rectangles is given by the set of coordinates of their corners. We will call this a purely geometrical description of a set of rectangles.

When an HIL description of a set of rectangles uses BR attributes they should often be consistently assigned. That means whenever a user specifies a BR attribute in a symbol definition it should always contain the minimum bounding rectangle which includes all rectangles (boxes) denoted by the symbol. Clearly, the minimum bounding rectangle for a set of rectangles denoted by a symbol in HIL can always be computed by following the hierarchic definition "bottom-up".

By imposing stronger constraints on the symbols which we might use to define new ones we can restrict the expressive power of HIL considerably. We may use the BR attributes (or other attributes) to state and check these constraints. As an example we will single out a restricted version of HIL which allows us only to define sets of rectangles which do not contain any pair of intersecting rectangles: Let us assume that every symbol definition is augmented with a consistent BR attribute. Then we require that all boxes and all BR's of the symbols occurring in Draw commands of a symbol definition do not intersect. Thus, under the assumption that all called symbols denote sets of nonintersecting rectangles the same holds true for the defined symbol. Let us call the thus restricted version of HIL consistent.

3. COMPLEXITY RESULTS

In this section we shall study the complexity of the various rectangle problems under the various formats for input. The primary results for this section are summarized in Table 1. In Subsection 3.1 we will review the results shown in the first column of Table 1; that section is just a survey of previous work In Subsection 3.2 we will study the results of the third column (in which the problems have unconstrained HIL input), and in Subsection 3.3 we will study the results of the second column (in which the problems have consistent HIL input).

Problem \ Input	Geometry Only	Consistent HIL	General HIL
1. Report Intersecting Pairs	N lg N + k	1	Exponential
2. Intersection Question	N lg N	1	NP-complete
3. Intersection with Query Object			
a. Point	N	N	NP-complete
b. Line	N	NP-complete	NP-complete
c. Rectangle	N	NP-complete	NP-complete
4. Northernmost Rectangle Below a Line	N	NP-complete	NP-complete
5. Measure Problem	N lg N	N	NP-complete
Perimeter Problem	N lg N	N	NP-complete
6. Connectedness Problems			
a. Single Component	(N+S) lg N	1	NP-hard
b. Number of Components	(N+S) lg N	N	NP-hard
c. Report all Components	N lg N + S·G(S)	Exponential	Exponential
7. Equality and Subset	N lg N	?	NP-hard
Testing of Rectangle Sets	N lg N	?	NP-hard

Table 1. Complexities of Rectangle Problems.

3.1 Geometric Input

In this subsection we shall review the complexity of rectangle problems when their inputs are given in geometric forms (or, equivalently, given as a list of Box-commands in HIL). Rectangle problems having this input format have recently been extensively studied in the literature. (See, for instance, Bentley and Wood [1980], van Leeuwen and Wood [1979], Vitanyi and Wood [1979], Nievergelt and Preparata [1980], and McCreight [1980].) We shall review the known results and sketch the basic techniques which have been used to obtain the results.

We will now consider the first problem in our list in some detail:

1. *Report Intersecting Pairs*

A naive algorithm checks all $\binom{N}{2}$ pairs of rectangles in a given set of N rectangles and thus solves the problem in quadratic time. This is optimal in the worst case because all N rectangles could intersect, yielding an output of size $\binom{N}{2}$. One can do better, however, by first sorting the 2N values of their lower and upper boundaries and then moving a scan line through the set bottom-to-top, keeping track of rectangles intersecting the current scan line. More precisely: Let us assume that every rectangle R is specified by the 4 values $(x_L(R), x_R(R), y_B(R), y_T(R))$ of their left, right, lower (bottom) and upper (top) boundaries. We sweep a horizontal scan line

SL through the set of rectangles. At each instant of time the scan line divides the set of rectangles into three disjoint sets: The set of dead rectangles which <u>have</u> been cut by SL, the set of active rectangles which <u>are</u> currently cut by the scan line SL, and the set of sleeping rectangles which <u>will</u> be cut by SL. These sets change only if SL passes a lower or upper boundary of some rectangle.

Whenever a sleeping rectangle becomes active, i.e. whenever SL halts at $y_b(R)$, we check all currently active rectangles for intersection with R. This strategy assures that we do not miss any pair of intersecting rectangles after sweeping SL once over the whole set of rectangles. Consider an instant of time, where a sleeping rectangle R becomes active, i.e. when SL halts at $y_b(R)$. How can we detect intersection with all currently active rectangles? Let us assume that $\bar{R}$ is such a rectangle. Then R and $\bar{R}$ intersect iff their projections to the x-axis overlap, i.e. iff $[x_l(R), x_r(R))$ and $[x_l(\bar{R}), x_r(\bar{R}))$ have a nonempty intersection. (Thus the scanning technique has reduced the intersection problem from a two-dimensional to a one-dimensional problem.)

Our above considerations show that it is sufficient to store the x-projections of the currently active rectangles in a data structure L such that we are able to answer the above question efficiently. Furthermore, L must be dynamically altered during the scan-line sweep: L is initially empty; whenever a sleeping rectangle R becomes active, its projection to the x-axis is inserted into L, and whenever R becomes dead, its projection to the x-axis is deleted from L.

The splitting of the one-dimensional overlapping segment problem into a range and inverse range query suggests to choose a pair of a range and a segment tree for L. Bentley and Wood [1980] used this to solve Problem 1 in time O(N log N + k), where k is the number of intersecting pairs, and in space O(N log N).

McCreight [1980] uses tile trees to improve that approach and obtain a solution with time complexity O(N log N + k) and space complexity O(N) which is optimal. Bentley, Haken and Hon [1980] use an array of segment bins for L which yields a linear expected time solution to the problem for sets of rectangles occurring in real chip designs.

We will now briefly examine the remaining rectangle problems when their input is given in geometric form.

2. *Intersection Question*

This problem can be solved by the same algorithm which was used to solve Problem 1: Just stop it after the first intersecting pair of rectangles was found, if there is one, or, if the scan line has passed the whole set of rectangles. Thus, the answer to question 2 can be computed in time O(N log N).

However, in order to detect intersection it is sufficient to keep the list of left and right boundaries of the currently active rectangles sorted according to their x-values in a simple AVL tree L during the scan line sweep bottom-to-top. Whenever a rectangle becomes active (respectively dead) its left and right boundaries are inserted into L (respectively deleted from L). There is at least one pair of intersecting rectangles in the set if and only if for at least one rectangle R either the newly inserted left or right boundary of R is squeezed in between the boundaries of any other active rectangle or the boundaries of R are separated by boundaries of any other active rectangle, i.e. the boundaries of R do not become immediate successors in L. This observation leads to a simpler O(N log N) time algorithm for solving Problem 2.

3. *Intersection with a Query Object*

It is obvious that we can detect intersection of a query object (a point, line, or rectangle) with at least one rectangle in a set of N rectangles in linear time. We can just use the naive approach of sequentially comparing each rectangle to the object and need no preprocessing. If preprocessing is allowed, it is possible to maintain the set of rectangles under sequences of insertions and deletions of rectangles; Vaishnavi and Wood [1980] claim to have a solution to the dynamic version of this problem which takes preprocessing time O(N log N) and query time O(log N).

4. *Northernmost Rectangle Below a Line*

Both the general and the special case of this problem can obviously be solved in linear time.

5. *Measure and Perimeter Problems*

These problems can be solved by the scan line technique which was used to solve Problems 1 and 2 above. When moving the scan line bottom-to-top through the set of rectangles we keep track of appropriate information about the currently active rectangles like the 1-dimensional measure of the projections of the currently active rectangles to the x-axis. Thus the measure and the perimeter can be accumulated in time O(N log N). See Bentley [1977], van Leeuwen and Wood [1979], and Vitanyi and Wood [1979] for the details.

6. *Connectedness Problems*

Nievergelt [1981] gives a solution for this problem which uses the scan line technique: The "dual graph" is constructed on-line during the sweep of the scan line for the even more general case where the given objects have arbitrary polygons as their boundaries. The "dual graph" reflects the connectedness structure of the objects in the set. If N and S are the total numbers of edges and intersections, respectively, the algorithm can be carried out in time O((N+S)logN). This implies that all three subproblems of Problem 6 can certainly be solved within the same time. It is left open whether or not an improvement to $O(N \log N)$ for the problems 6a) and b) and to $O(N \log N + k)$ for problem 6c) is possible, where k denotes the number of connected components.

However, it is easy to see that one can solve the connectedness problems in time $O(N \log N + S \cdot G(S))$, for a set of N rectangles with S intersections, where G denotes the inverse of the Ackermann function. For, the scan line technique reduces the problem to the problem of determining pairwise intersections and performing at most S Union or Find operations.

7. *Equality and Subset Testing of Rectangle Sets*

By first sorting the two sets of coordinates we obviously can get a solution to these problems in time $O(N \log N)$, using the methods used in Problems 1 and 5.

3.2 General HIL Input

In this subsection we shall study the complexity of rectangle problems when their inputs are given in the Hierarchical Input Language HIL. Most of the results that we will see in this subsection are negative; that is, we will see that most of the problems either provably require exponential time (because an extremely concise HIL description can generate exponentially large output) or are NP-complete (because HIL can generate rectangle sets in which NP-hard problems can be encoded). Because all of our proofs use only two primary constructions (giving an exponentially large output and encoding an NP-hard problem), we will first examine those two constructs in detail in Subsection 3.2.1, and then turn to the rest of the problems in Subsection 3.2.2.

3.2.1 Pairwise Intersection Problems

In this section we shall study two problems that concern the pairwise intersections among a set of rectangles specified by an HIL description of length N (recall that the length of an HIL description is defined to be the total number of Box and Draw commands contained in the input). The two problems are to report all intersecting pairs of rectangles, and to determine whether any two pairs intersect. We shall first examine the problem of reporting all intersecting pairs, which the following theorem shows is difficult in the worst case.

Theorem 1:

The problem of reporting all intersecting pairs of rectangles defined by an HIL description of length N must sometimes require time exponential in N.

Proof:

We will construct a particular HIL file of length N that contains $2^{N/2}$ overlapping unit squares with lower-left corners at the origin; because the output must include all $\binom{2^{N/2}}{2}$ pairs, it is of size $(2^{N/2})\cdot(2^{N/2}-1)/2$, or approximately 2^{N-1}. Symbol 1 in this file is defined as

```
1: BR (0,0),(1,1)
   Box (0,0),(1,1)
   Box (0,0),(1,1),
```

so it consists of two unit squares with lower-left corners at the origin. The ith symbol, for $2 \le i \le N$, is defined as

```
i: BR (0,0),(1,1)
   Draw i-1 at (0,0)
   Draw i-1 at (0,0).
```

It is easy to prove by induction that the ith symbol contains exactly 2^i overlapping rectangles, so the Nth rectangle contains 2^N, and the construction is complete. QED.

We will now turn our attention to the more subtle problem of testing whether any two elements intersect in a set of rectangles given by an HIL of length N. Our primary result for this problem is that it is NP-complete; our first step toward showing this is the following lemma.

Lemma 2:

The HIL intersection question is in NP.

Proof:

The nondeterministic algorithm first guesses two rectangles, then verifies that they intersect. A rectangle is guessed by starting at the root symbol and nondeterministically following down the call structure of the HIL description until a Box command is reached. QED.

Note that the above proof is concise precisely because a certificate of intersection for a particular HIL is so simple: we merely display the two intersecting rectangles.

The next part of the proof is the more substantial: we will demonstrate that the HIL intersection question is in fact NP-hard. The reduction is to the knapsack problem, which asks whether there is some subset of a set of integers whose sum is a given integer (see Garey and Johnson [1979]).

Lemma 3:

The HIL intersection question is NP-hard.

Proof:

We will show that the question of whether some subset of the set of positive integers $W = \{w_1, w_2, \ldots, w_N\}$ sums to the given integer T can be reduced in polynomial time to an HIL intersection problem. Our first step is to define in N+1 HIL symbols a set of 2^N x-by-x rectangles (for any $0<x<1$, say $x=1/2$)* whose left hand sides are aligned along the integers from 0 to 2^N-1 and whose bottom sides have heights corresponding to the sums of all 2^N subsets of weights. Symbol 0 is defined as

```
0: BR (0,0),(x,x)
   Box (0,0),(x,x)
```

* Note that we have taken a liberty with the definition of HIL by using a rectangle of the noninteger size (1/2,1/2). This could easily be fixed, but that would only obscure the structure of the proof.

For $1 \le i \le N$, symbol i is defined as

```
    i: BR(0,0),($2^i$-1+x, x+ $\sum_{1 \le j \le i} w_j$)
          Draw i-1 at (0,0)
          Draw i-1 at ($2^{i-1}$,$w_i$).
```

Note that the heights of the bottoms of the rectangles in symbol i represent the sums of all the subsets of $\{w_1, w_2, \ldots, w_i\}$; this is easily proved by induction.

Now that we have represented the sums of all the subsets by a sequence of rectangles at various heigths, we must do the same for the desired sum T. Symbol N+1 is defined as

```
    N+1: BR(0,T),(1+x,x)
          Draw 0 at (0,T)
          Draw 0 at (1,T);
```

it places two x-by-x rectangles at height T. We then copy those rectangles by defining, for $2 \le i \le N$,

```
    N+i: BR(0,T),($2^i$-1+x,x)
          Draw N+i-1 at (0,0)
          Draw N+i-1 at ($2^{i-1}$,0).
```

Note that the symbol N+N = 2N consists of a row of 2^N rectangles with bottoms at height T and left sides along the integers from 0 to 2^N-1.

The stage is now completely set; the final symbol is defined as

```
    2N+1: BR(0,0)($2^N$-1+x, x+ $\sum_{1 \le i \le N} w_i$)
          Draw N at (0,0)
          Draw 2N at (0,0)
```

and there is a solution to the knapsack problem if and only if some pair of rectangles in symbol 2N+1 intersect (for by the distinctness of x-values in symbols N and 2N, two rectangles intersect if and only if they share the same y-value of T). QED.

An example of the construction used in this proof is given as Example 1. The two above lemmas can now be combined to prove Theorem 4.

Theorem 4:

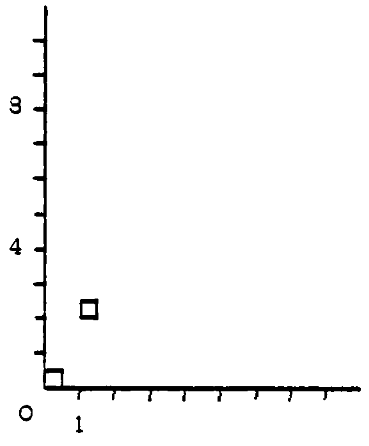
The HIL intersection question is NP-complete.

Proof:

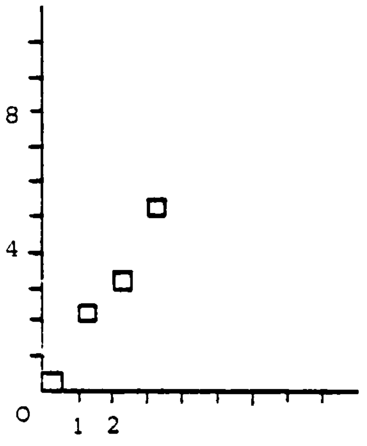
Immediate from Lemmas 2 and 3.

Example 1

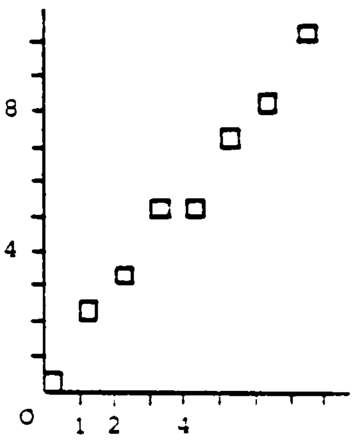
In this example we will see how a particular instance of the knapsack problem can be reduced to a problem of testing for intersection in a rectangle set defined in HIL. We will assume that the set W is {2,3,5} and we want to know whether any subset sums to 5. Symbols 1,2,3 are illustrated below.



Symbol 1

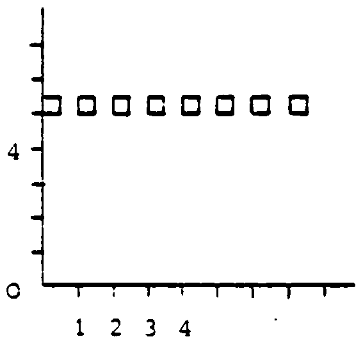


Symbol 2



Symbol 3

Symbol 6 has the structure



When symbols 3 and 6 are overlayed they have two intersections, at (3,5) and (4,5); these correspond to the subsets {2,3} and {5}, both of which sum to 5. End of Example.

Because we will make extensive use of the construction used in the proof of Lemma 3, it is important that we analyze the construction in some detail. The first aspect to note is that because we reduced the HIL intersection question to the knapsack problem, which is known to be solvable in pseudo-polynominal time, we have shown only the weak NP-completeness (and not the strong NP-completeness) of the HIL intersection question*. This is an essential aspect of the proof, because the HIL intersection question is itself solvable in pseudo-polynomial time (that is, it is solvable in polynomial time if the inputs are expressed in unary). To prove this it suffices to observe that both the height and width of the minimum bounding rectangle of the root symbol are less than the sum of all the input parameters; thus all the symbols in the set must be placed on an integer grid of size at most the square of the input length. We can record for each cell in the grid which of the linear number of symbols have already been instantiated there, and thus avoid doing more than a polynomial amount of checking.

* For a discussion of the notion of weak NP-completeness, see Garey and Johnson [1979].

The second important fact to note is that the HIL descriptions constructed in the proof of Lemma 3 are in a very well-behaved subset of HIL. The most important property is that 2N+1 of the 2N+2 symbols are consistent in the sense that the bounding boxes of their symbols do not intersect; only symbol 2N+1 has intersecting subsymbols. Thus the HIL intersection question remains NP-complete even when we constrain the HIL input to contain at most one pair of overlapping called symbols. There are a number of other senses in which the graph of the construction is well-behaved: it uses only one Box command, every symbol (except 0) makes exactly two calls, and every symbol (except 0) is called exactly twice by exactly one other symbol.

3.2.2 Other Problems with HIL Input

In the previous subsection we saw the following results for rectangle problems with inputs in HIL.

1. *Report Intersecting Pairs*

   Theorem 1 showed by a counting argument that this problem must require exponential time in the worst case.

2. *Intersection Question*

   Theorem 4 showed that this problem is (weakly) NP-complete, using a reduction to the knapsack problem.

We will now briefly examine the remaining rectangle questions and problems when their inputs are given in HIL.

3. *Intersection with a Query Object*

   We will consider the three subproblems in decreasing order of generality.

   c) *Intersection with a Rectangle*

   To prove this problem NP-complete we will use the construction in the proof of Lemma 3 and replace the 2^N squares at height T with one rectangle of height x and width 2^N-1+x, with bottom left corner at (O,T). The knapsack problem has a solution if and only if that rectangle intersects any other.

   b) *Intersection with a Line*

   Replace the query rectangle in the above construction by a horizontal line with height T+x/2.

   a) *Intersection with a Point*

   Use the construction of parts a and b, but place all squares to have their left edges on the line x=O. The query point at (x/2,T+x/2) intersects a rectangle if and only if the knapsack problem has a solution.

4. *Northernmost Rectangle Below a Line*

The general problem can be proved NP-complete by the proof of Problem 3b; if the northernmost rectangle below the line $y = T+1-\epsilon$ is at $y=T$, then and only then does the knapsack problem have a solution. The special case of finding the northernmost rectangle in the entire set is solvable in linear time by computing "bottom-up" the northernmost rectangle in every symbol.

5. *Measure and Perimeter Problems*

To prove these problems NP-complete we will use the construction in the proof of 3a, in which the sums of all subsets are represented by a set of rectangles at a vertical line. We first construct such a set and compute its measure, then augment the old set by a single x-by-x square at height T; the new measure is x^2 greater than the old if and only if there is no solution to the knapsack problem. Likewise, the perimeter increases by 4x.

6. *Connectedness Problems*

a) *Are the Rectangles in a Single Connected Component?*

We will show that this problem is NP-hard by using the same constructions as for problems 3a und 5. We first place all squares with their bottom sides at the sums of all subsets and their left sides along the y-axis. We then place a rectangle of width 1 and height equal to x plus the sum of all weights in W to have its right side along the y-axis. Note that at this time all of the rectangles form a single connected component. Finally, we place a single (x/2,x) rectangle at (x/2,T); that rectangle is in the single connected component if and only if the knapsack problem has a solution. Note that this proof shows only that the problem is NP-hard, and not that it is in NP.

b) *Number of Connected Components*

By the reduction of 6a, this problem is also NP-hard.

c) *Enumeration of All Connected Components*

The construction used in the proof of Theorem 1 can be slightly augmented to produce in N symbols a total of 2^N disjoint squares; the time to output them alone shows that this problem must require exponential time.

7. *Equality and Subset Testing of Rectangle Sets*

The construction used in question 6a shows that these problems are NP-hard.

3.3 Consistent HIL Input

In this subsection we shall study the complexity of rectangle problems when their inputs are given as consistent HIL descriptions. That is, for every symbol in the HIL description, the bounding boxes of all objects within the symbol must be contained within the symbol's bounding box and nonoverlapping. This restriction has very different impacts on the complexity of the various rectangle problems: whereas most of the problems with general HIL input were NP-hard, we will see that for consistent designs some problems remain NP-hard, while other problems are now solvable in polynomial time (usually linear or $\Theta(N \lg N)$), and still others become trivial (that is, they can be solved in constant time).

The first problem that we must face when dealing with consistent designs is that of verifying that an allegedly consistent design does in fact satisfy the properties of having only contained and nonoverlapping subsymbols. This property is easy to verify using the scanning algorithm to solve Problem 2 in Subsection 3.1 (that is, given N rectangles in the plane, do any intersect?); recall that $\Theta(N \lg N)$ time is necessary and sufficient to test this property. Our algorithm for testing consistency will now proceed bottom-up through the HIL design, symbol-by-symbol, and use the geometry-only intersection checker to verify that no subsymbols in any symbol intersect; it is trivial to ensure in linear time that all subsymbols are in fact contained within the symbol's bounding rectangle. By the fact that there are a total of O(N) rectangles and calls on subsymbols altogether in the HIL and the fact that the function $\Theta(N \lg N)$ is concave upward, the entire algorithm takes at most $O(N \lg N)$ time. Note that $\Theta(N \lg N)$ is necessary for the case of a one-level consistent design, so this bound is best possible.

We turn now to study the rectangle problems themselves. The first two problems are trivial for designs that we know to be consistent.

1. *Report Intersecting Pairs*
2. *Intersection Question*

   Both of these problems can be answered in constant time because a consistent design is known to have no intersecting pairs of rectangles.

The remaining problems do not admit trivial solutions.

3. *Intersection with a Query Object*

   We will consider the three subproblems in increasing order of generality.

   a) *Intersection with a Point*

      This problem can be solved in linear time. To see if any of the symbols intersects a given point we start at the root symbol and then recursively search down the DAG that is the HIL description; at each symbol we visit at most one of its subsymbols. (Note that if we have to visit more, then the subsymbols overlap, which violates consistency).

b) *Intersection with a Line*

c) *Intersection with a Rectangle*

Because the HIL descriptions used in Subsection 3.2 to prove the NP-hardness of these questions for general designs were in fact consistent, both problems b and c remain NP-complete for consistent designs.

4. *Northernmost Rectangle Below a Line*

Because the proof in Subsection 3.2 uses a consistent design, this problem remains NP-complete.

5. *Measure and Perimeter Problems*

These problems are both solvable in linear time. To solve the measure problem we proceed bottom-up through the set, computing for each symbol the sum of the measures of the rectangles it contains by adding together the (previously calculated) measures of the subrectangles it calls. The perimeter problem is solved in a similar fashion.

6. *Connectedness Problems*

a) *Are the Rectangles in a Single Connected Component?*

The answer is yes if and only if there is exactly one rectangle in the set.

b) *Number of Connected Components*

This can be solved in linear time by a bottom-up algorithm like the algorithms used to solve problem 5.

c) *Enumeration of All Connected Components*

The construction used in Subsection 2.2 to show that this problem can require exponential time still holds.

7. *Equality and Subset Testing of Rectangle Sets*

We leave these as open problems.

4. IMPLICATIONS AND OPEN PROBLEMS

The purpose of this paper has been to lay a solid theoretical foundation for the manipulation of hierarchically defined sets of rectangles in the plane. There are two motivations for this approach: the questions are interesting from a purely mathematical viewpoint, and the theory can occasionally have a substantial impact on practice. For instance, the asymptotic worst-case rectangle intersection algorithm of Bentley and Wood [1980] motivated the efficient expected-time algorithm of Bentley, Haken and Hon [1980], which was in turn used in the VLSI Design Rule Checker of Haken [1980].

The primary results of this paper are summarized in Table 1. The first column of that table surveys previous work on geometrically defined rectangle problems, the third column shows that most problems are NP-hard when presented with unrestricted HIL inputs, and the second column shows that when the designs are constrained to be consistent, then most of the problems become rather easy to solve. These facts correspond closely to the experience of Hon [1980] and Whitney [1980] in using their programs that manipulate hierarchical VLSI designs: highly structured designs (which are never consistent but usually rather close in some sense) can be processed very quickly, while highly unstructured designs require prohibitive amounts of processing time.

It is important to state carefully the implications of the above results for the builders of systems that process hierarchical designs. The NP-hardness results do not state that such designs cannot be processed efficiently; rather, they imply that it is highly unlikely that one can ever find an algorithm that will efficiently process every design. Thus one should not search for such an algorithm, but rather focus one's energy on algorithms that work well for an important subclass of designs.

There are two types of subclasses that might be investigated, and both appear to offer much to theoretician and practitioner alike.

1. *A Statistical Approach*

Using this approach one would first build a probabilistic model of VLSI designs, and then design an algorithm that performs well on the average when the inputs are drawn from that distribution. (This is the approach taken for the geometry-only rectangle intersection problem by Bentley, Haken and Hon [1980]). Devising a probabilistic model that includes both the graph-theoretic aspects of the HIL structure and the geometric aspects of the shapes and placement of the rectangles is a subtle mathematical problem; fitting such a model to actual data will require an exceptionally talented practitioner.

2. *A Restriction Approach*

We saw that restricting the designs to be consistent allowed many of the problems to be solved quite efficiently. Unfortunately, consistency is so restrictive that no real designs can be built using it! We therefore observe a tradeoff between severe restrictions (which exclude many designs but facilitate rapid processing) and lax restrictions (which exclude few designs but admit many that are very time consuming to process). It will be important to identify families of restrictions that exclude only a few designs (and hopefully uninteresting ones at that) but admit to very rapid processing of the remaining designs.

The NP-completeness results of Subsection 3.2 have a rather interesting implication for this endeavor. Recall that the Rectangle Intersection Question is NP-complete when the inputs are presented in HIL. Many people suspect that this implies that the complement of the problem is not even in NP (see, for instance, Garey and Johnson [1979, Section 7]). This in turn would imply that there can never be a polynomial-length certificate of nonintersection for a set of rectangles. This means that if a restriction approach is taken in which the designer adds a polynomial amount of extra information and the resulting design can be processed in polynomial time, then some valid designs must necessarily have been excluded.

Bibliography

Bentley, J.L. [1977]: Solution to Klee's rectangle problems, unpublished manuscript, Dept. of Computer Science, Carnegie-Mellon University, 1977.

Bentley, J.L. and Wood, D. [1980]:An optimal worst-case algorithm for reporting intersections of rectangles, IEEE Transactions on Computers, Vol. C-29, 1980, 571-577.

Bentley, J.L., Haken, D., and Hon, R. [1980]: Statistics on VLSI Designs, Carnegie-Mellon University, Computer Science Department, Technical Report CMU-CS-80-111.

Garey, M.R. and Johnson, D.S. [1979]: Computers and Intractability, A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979

Haken, D. [1980]: A geometric design rule checker, VLSI Document V053, Carnegie-Mellon University, 9 June 1980, 9pp.

Hon, R. [1980]: The Hierarchical Analysis of VLSI Designs, Thesis proposal, Carnegie-Mellon University, Dec. 1980

Klee, V. [1977]: Can the Measure of $U[a_i,b_i]$ be computed in less than $O(n \log n)$ steps, Research Probl. Sect., Amer. Math. Monthly 84, 1977, 284-285.

Lauther [1980]: A Data Structure for Gridless Routing, 17th Design Automation Conference, Minneapolis, 1980, 1-7.

van Leeuwen, J. and Wood, D. [1979]: The Measure Problem for Rectangular Ranges in d-Space, Technical Report, RUU-CS-79-6, July 1979.

Mead, C. and Conway, L. [1980]:Introduction to VLSI Systems, Addison-Wesley.

Nievergelt, J. and Preparata, F.P. [1980]: Plane-sweep algorithms for intersecting geometric figures, Technical Report (in preparation), Institut für Informatik, ETH, Zürich.

Nievergelt, J. [1981]: Private Communication.

Vaishnavi, V. and Wood, D. [1980]: Rectilinear line segment intersection, layered segment trees and dynamization, Computer Science Technical Report, 80-CS-8, McMaster University, Hamilton, Ontario, Canada.

Vitanyi, P.M.B. and Wood, D. [1979]: Computing the Perimeter of a Set of Rectangles , Computer Science Technical Report, 79-CS-23, McMaster University, Hamilton, Ontario, Canada.

Whitney, T. [1980]: Description of the Hierarchical Design Rule Filter, Caltech SSP File 4027, Oct. 1980.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CMU-CS-81-109	2. GOVT ACCESSION NO. AD A106552	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE COMPLEXITY OF MANIPULATING HEIRARCHICALLY DEFINED SETS OF RECTANGLES		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) JON LOUIS BENTLEY THOMAS OTTMANN		8. CONTRACT OR GRANT NUMBER(s) N00014-76-C-0370
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Computer Science Department Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlington, VA 22217		12. REPORT DATE APRIL 1981
		13. NUMBER OF PAGES 41
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DATE
ILME